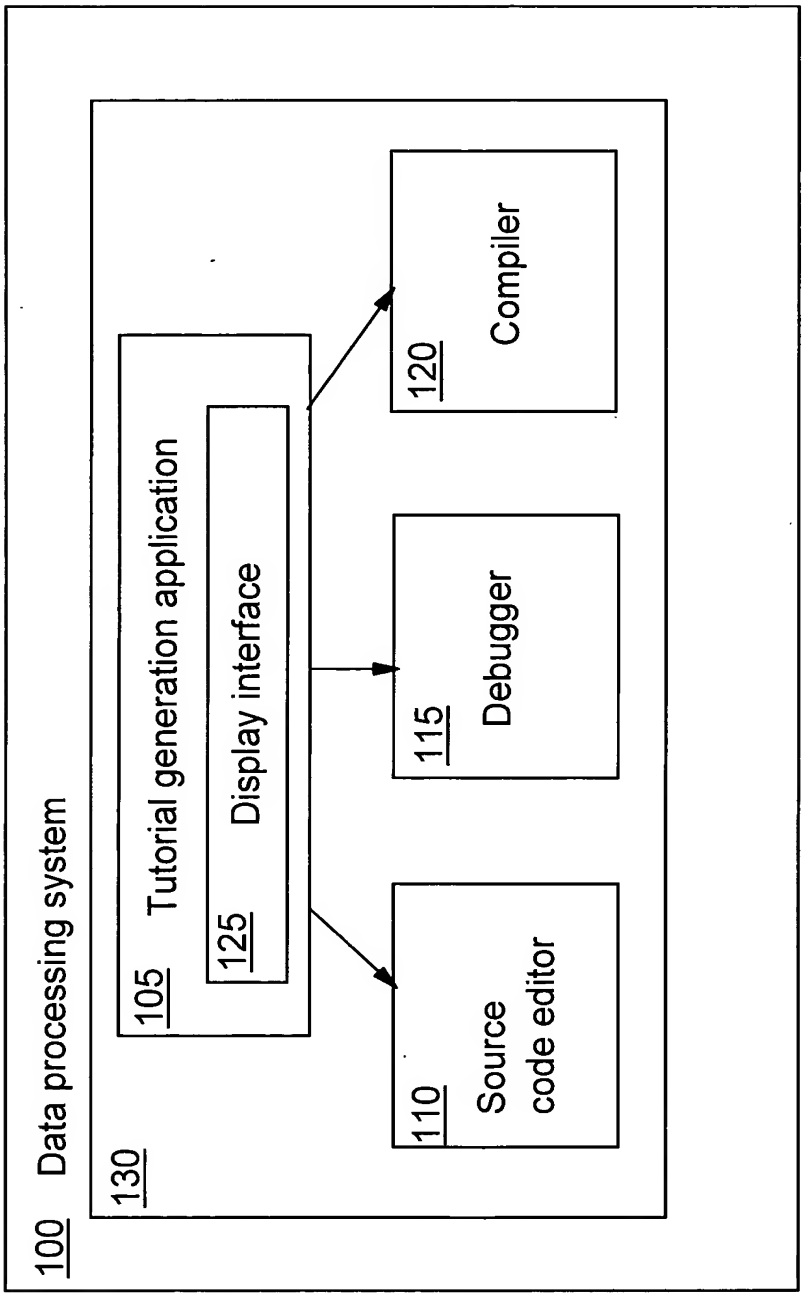
Fig 1



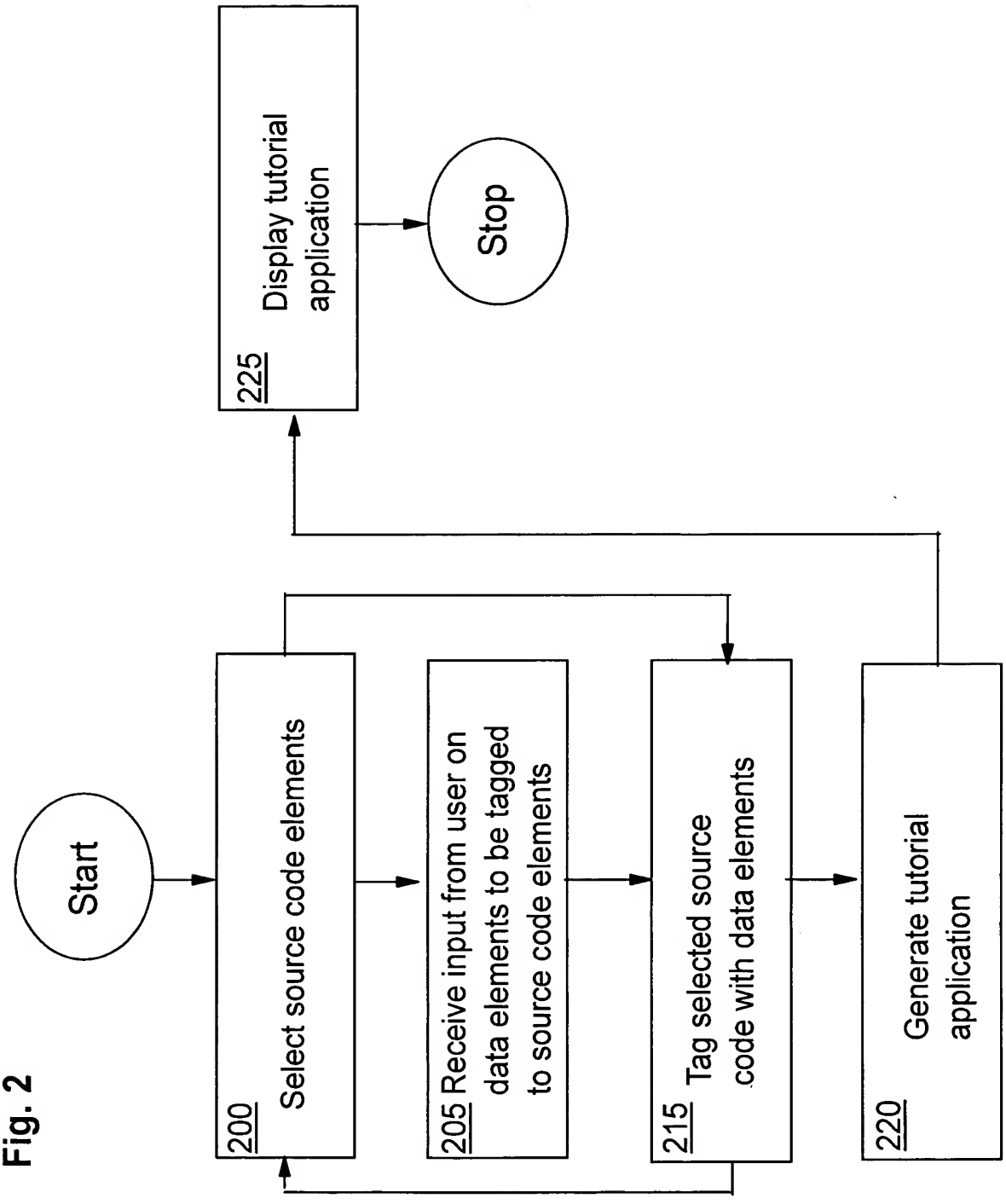

Start
200 Select source code elements
205 Receive input from user on data elements to be tagged to source code elements
215 Tag selected source code with data elements
220 Generate tutorial application
225 Display tutorial application
Stop

Fig. 2

Fig. 3

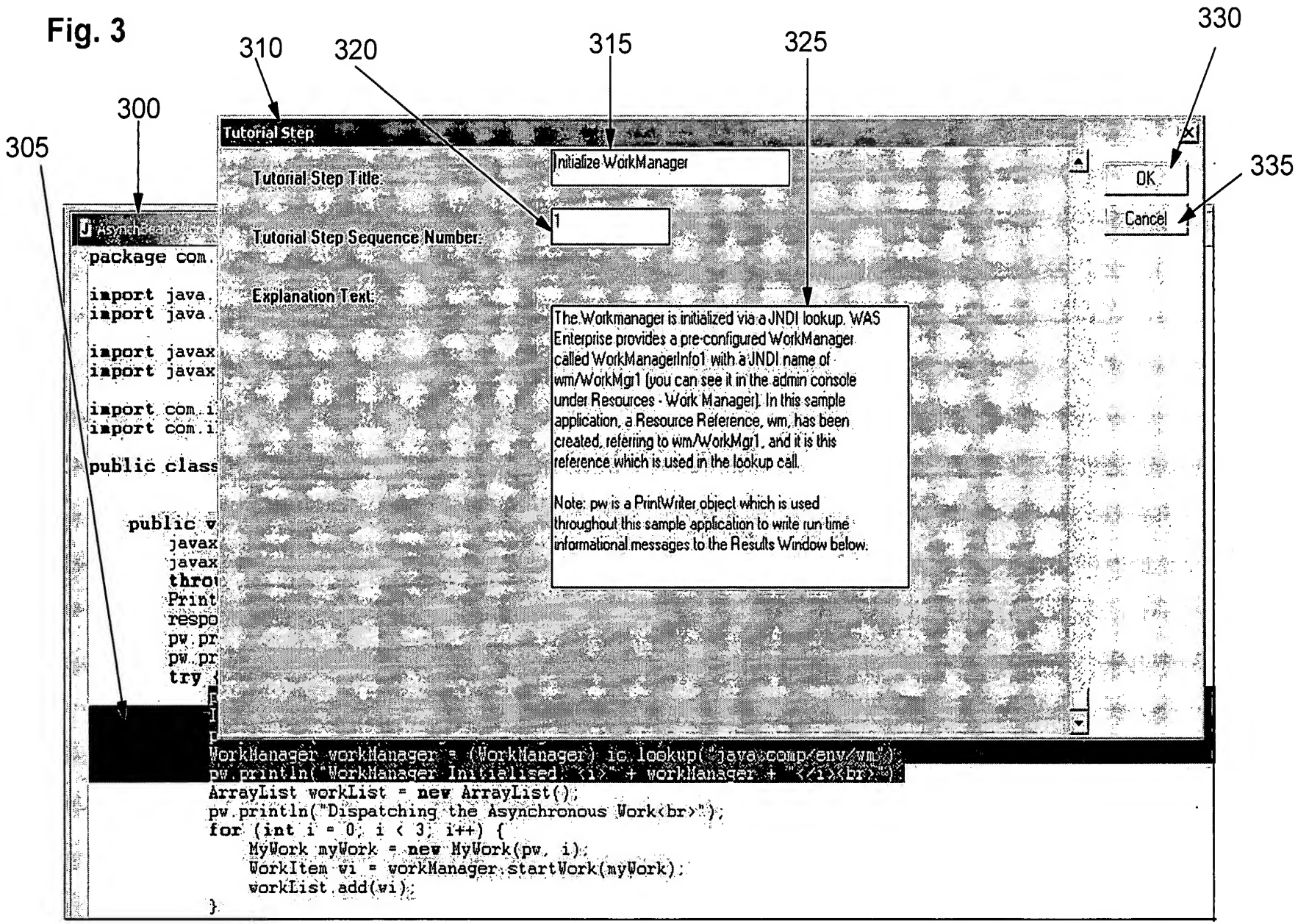

310
320
315
325
330
300
305
335
Tutorial Step
Tutorial Step Title:
Initialize WorkManager
Tutorial Step Sequence Number:
1
Explanation Text:
The Workmanager is initialized via a JNDI lookup. WAS Enterprise provides a pre-configured WorkManager called WorkManagerInfo1 with a JNDI name of wm/WorkMgr1 (you can see it in the admin console under Resources - Work Manager). In this sample application, a Resource Reference, wm, has been created, referring to wm/WorkMgr1, and it is this reference which is used in the lookup call.
Note: pw is a PrintWriter object which is used throughout this sample application to write run time informational messages to the Results Window below.
OK
Cancel
package com.
import java.
import java.
import javax
import javax
import com.i
import com.i
public class
public v
javax
javax
throw
Print
respo
pw.pr
pw.pr
try
WorkManager workManager = (WorkManager) ic.lookup("java:comp/env/wm");
pw.println("WorkManager Initialised: " + workManager + "
");
ArrayList workList = new ArrayList();
pw.println("Dispatching the Asynchronous Work
");
for (int i = 0; i < 3; i++) {
MyWork myWork = new MyWork(pw, i);
WorkItem wi = workManager.startWork(myWork);
workList.add(wi);
}

Fig. 4

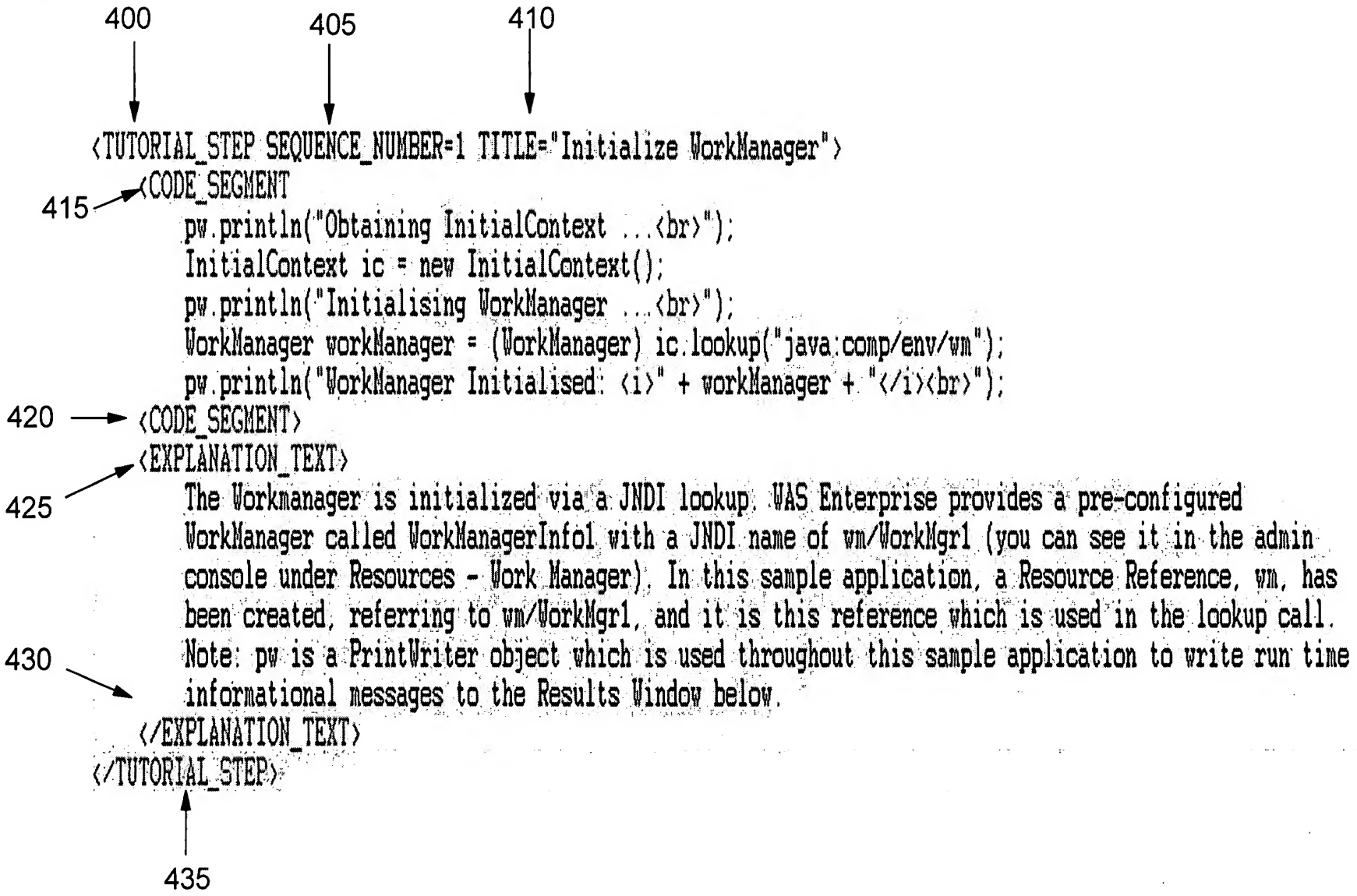
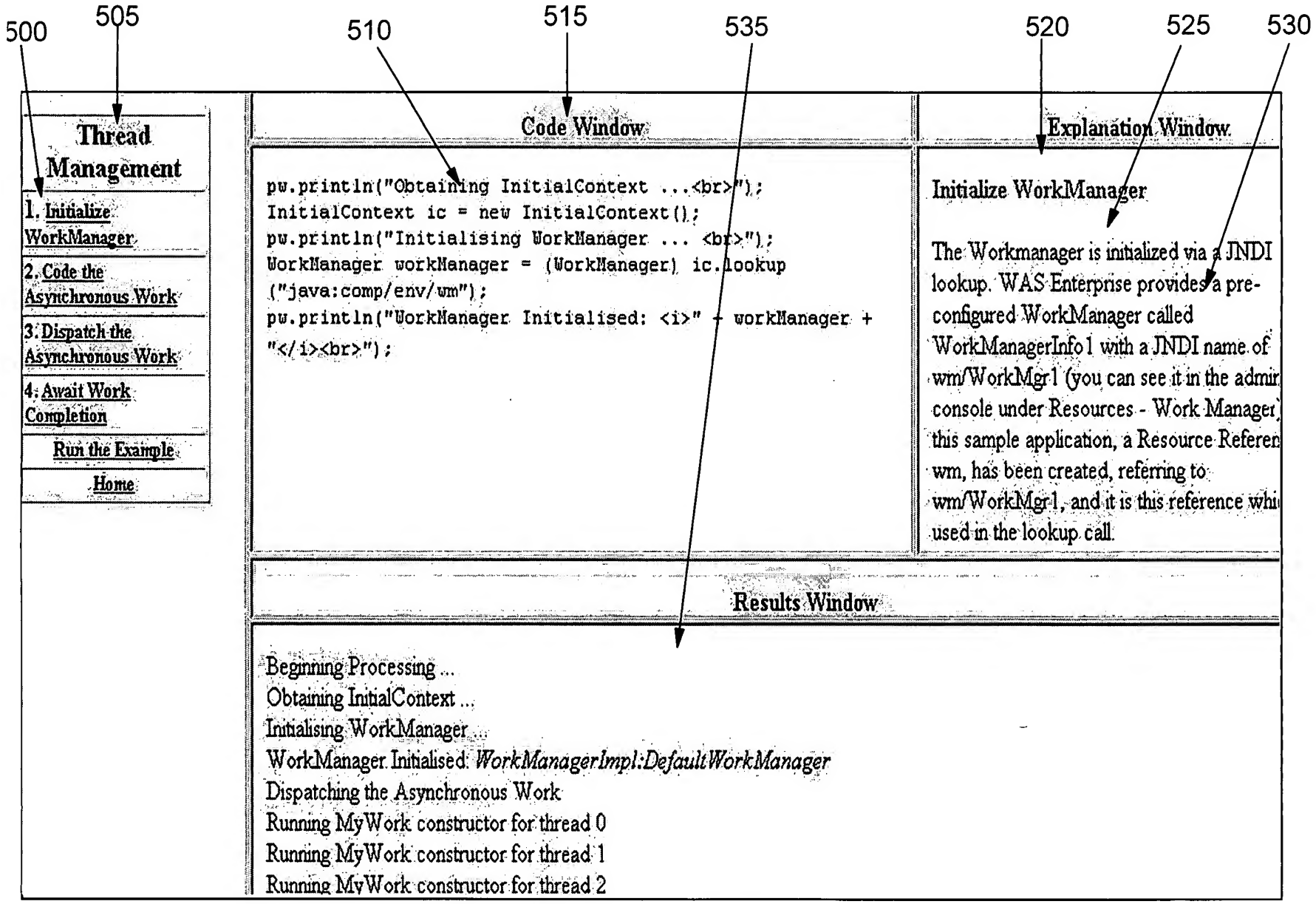

400
405
410
415
420
425
430
435
<TUTORIAL_STEP SEQUENCE_NUMBER=1 TITLE="Initialize WorkManager">
<CODE_SEGMENT
pw.println("Obtaining InitialContext ...
");
InitialContext ic = new InitialContext();
pw.println("Initialising WorkManager ...
");
WorkManager workManager = (WorkManager) ic.lookup("java:comp/env/wm");
pw.println("WorkManager Initialised: " + workManager + "
");
<CODE_SEGMENT>
<EXPLANATION_TEXT>
The Workmanager is initialized via a JNDI lookup. WAS Enterprise provides a pre-configured
WorkManager called WorkManagerInfo1 with a JNDI name of wm/WorkMgr1 (you can see it in the admin
console under Resources - Work Manager). In this sample application, a Resource Reference, wm, has
been created, referring to wm/WorkMgr1, and it is this reference which is used in the lookup call.
Note: pw is a PrintWriter object which is used throughout this sample application to write run time
informational messages to the Results Window below.
</EXPLANATION_TEXT>
</TUTORIAL_STEP>

Fig. 5


500
505
510
515
535
520
525
530
Thread Management
1. Initialize WorkManager
2. Code the Asynchronous Work
3. Dispatch the Asynchronous Work
4. Await Work Completion
Run the Example
Home
Code Window
pw.println("Obtaining InitialContext ...
");
InitialContext ic = new InitialContext();
pw.println("Initialising WorkManager ...
");
WorkManager workManager = (WorkManager) ic.lookup
("java:comp/env/wm");
pw.println("WorkManager Initialised: " + workManager +
"
");
Explanation Window
Initialize WorkManager
The Workmanager is initialized via a JNDI lookup. WAS Enterprise provides a pre-configured WorkManager called WorkManagerInfo1 with a JNDI name of wm/WorkMgr1 (you can see it in the admir console under Resources - Work Manager) this sample application, a Resource Referen wm, has been created, referring to wm/WorkMgr1, and it is this reference whi used in the lookup call.
Results Window
Beginning Processing ...
Obtaining InitialContext ...
Initialising WorkManager ...
WorkManager Initialised: WorkManagerImpl:DefaultWorkManager
Dispatching the Asynchronous Work
Running MyWork constructor for thread 0
Running MyWork constructor for thread 1
Running MyWork constructor for thread 2